Komunikacja sieciowa w rozwiązaniu OpenSSI

Projekt z przedmiotu RSO

Michał Grabowski
M.Grabowski@elka.pw.edu.pl

16 czerwca 2005

Spis treści

1 Wprowadzenie

Zagadnienie komunikacji sieciowej w rozwiązaniu OpenSSI można podzielić na dwie odrębne, choć często przeplatające się części:

- komunikację wewnętrzną,
- komunikację standardową.

Komunikacja wewnętrzna (ang. *interconnect communication*) jest związana z niskopoziomowym połączeniem pomiędzy węzłami. Jądra poszczególnych węzłów klastra używają dedykowanych połączeń do realizowania idei SSI.

W komunikacji standardowej ważne jest zapewnienie widoczności klastra jako pojedynczej, wysoko dostępnej (ang. *highly available*) maszyny. W związku z tym tworzone są wirtualne adresy IP klastra (CVIP) umieszczone w sieci zewnętrznej.

2 Komunikacja wewnętrzna

Aby idea SSI była realizowalna i klaster mógł działać, każdy węzeł musi być wyposażony w co najmniej jeden interfejs sieciowy, który będzie służył do komunikacji pomiędzy jądrami poszczególnych węzłów klastra. Komunikacja taka jest całkowicie

wewnętrzna i nie jest widoczna dla świata zewnętrznego. Używany jest w tym celu zwykle interfejs ethernetowy. Oznacza to jednak, że obsługa sieci na takim interfejsie musi zostać włączona bardzo wcześnie w procesie ładowania systemu (tzn. w dysku wirtualnym (*ramdisk*), zanim zostanie zamontowany korzeń (ang. *root*) tak, aby węzeł mógł uczestniczyć w tworzeniu klastra i procesie prowadzącym do decyzji kto powinien zamontować korzeń systemu plików. Z tego powodu konfigurowanie i modyfikowanie takiego interfejsu różni się od działań związanych z innymi interfejsami sieciowymi.

Dla każdego węzła należy wybrać jeden interfejs na połączenie wewnętrzne (ang. *interconnect*). Nazwa odpowiadająca adresowi IP tego interfejsu jest wstawiana do `/etc/nodename` (który jest symbolicznym połączeniem do `/cluster/node/etc/nodename`) w trakcie procesu instalacji oraz przez skrypty `openssi-config-node`. Nazwa ta stanie się również nazwą hosta (ang. *hostname*) dla tego węzła (poprzez plik konfiguracyjny `/etc/sysconfig/network`). W pliku `/etc/clustertab` znajdują się adresy fizyczne MAC oraz adresy sieciowe IP interfejsów komunikacji wewnętrznej wszystkich węzłów klastra. Plik ten jest tworzony i zarządzany podczas instalacji oraz skrypty `openssi-config-node`, a informacje w nim zawarte służą `mkinitrd` do stworzenia pliku `/etc/boottab` w ramdisku, z którego korzystają wszystkie węzły podczas uruchamiania. Inne miejsca zawierające istotne dane to:

- `/etc/hosts` - może zawierać adresy i nazwy węzłów (podobnie jak zewnętrzny serwer DNS),
- `/etc/dhcpd.conf` - może zawierać adresy MAC i IP. Ten plik jest generowany automatycznie przez `mkinitrd`.

Aby programistycznie określić który interfejs jest wykorzystywany do komunikacji wewnętrznej należy wykorzystać fukcję z biblioteki *libcluster* o nazwie clusternode_get_ip, która zwróci adres IP dowolnego węzła o podanym numerze. Często jest ważne żeby wszystkie węzły miały drogę (ang. *route*) do świata na zewnątrz klastra. Dla węzła początkowego, taka droga prawdopodobnie zostanie ustawiona wcześniej, jeszcze przed instalacją OpenSSI. W trakcie, gdy nowe węzły są dodawane do klastra, ważne jest aby każdy węzeł miał dostęp do bramki (ang. *gateway*).

Aby sprawdzić i potwierdzić konfigurację, można użyć kilku narzędzi, których wynik może być przydatny w razie problemów:

- `cat /etc/clustertab` pokaże adresy IP wszystkich węzłów odpowiadające interfejsom wykorzystywanym do komunikacji wewnętrznej,
- `ifconfig` - na każdym węźle pokaże adresy IP skojarzone z poszczególnymi interfejsami węzła,
- `host <adresIP>` - dla adresów IP kolejnych węzłów wyświetlona zostanie odpowiadająca im nazwa z `/etc/nodename`,
- `onall cat /etc/nodename` - poda unikalne nazwy wszystkich węzłów,
- `/etc/dhcpd.conf` - jeśli węzły w klastrze ustawione są na uruchamianie PXE (a nie Etherboot), wtedy w tym pliku będzie można znaleźć adresy MAC i IP węzłów (podobnie jak w `/etc/clustertab`),
- `onnode -p # route` lub `onnode -p # netstat -r` - sprawdzenie tablicy routingu węzła o numerze #.

Zarządzanie interfejsem komunikacji wewnętrznej klastra ma w docelowej wersji OpenSSI odbywać się przez opcję zmiany węzła skryptu `openssi-config-node`. Na chwilę obecną konfigurację należy przeprowadzać ręcznie. Oto lista kilku zadań, które można samemu przeprowadzić:

- **zmiana adresu IP interfejsu komunikacji wewnętrznej**
  Aby zmienić adres IP interfejsu komunikacji wewnętrznej należy wyedytować plik `/etc/clustertab`, uruchomić `mkinitrd` (ze wszystkimi wymaganymi parametrami: `mkinitrd --tablonly -/boot/initrd-xxx` gdzie xxx oznacza nazwę jądra ssi, a następnie uruchomić `ssi-ksync` (aby przenieść jądro/ramdisk na wszyskie partycje rozruchowe). Jeśli potrzeba, nalezy wyedytować plik `/etc/hosts` i uaktualnić odpowiednie wpisy DNS. Dobrym pomysłem jest również poprawienie pliku `/cluster/node#/etc/sysconfig/network-scripts/ifcfg-eth#` jeśli istnieje. Następnie należy zrestartować system. Uwaga: warto włączyć wszystkie węzły, które mają lokalne partycje rozruchowe podczas uruchamiania `ssi-ksync`.

- **zmiana nazwy interfejsu komunikacji wewnętrznej (nazwa węzła/hosta)**
  Aby dokonać zmiany nazwy interfejsu, wystarczy wyedytować plik `/cluster/node#/etc/nodename` dla węzła o numerze #. Jeśli potrzeba, należy również uaktualnić dane w pliku `/etc/hosts` oraz wpisy w DNS. Należy zrestartować węzeł.

- **zmiana adresu MAC interfejsu komunikacji wewnętrznej (wymiana karty sieciowej w węźle)**
  Postępowanie podobnie jak w przypadku zmiany adresu IP. Należy pamiętać, że wszystkie interfejsy komunikacji wewnętrznej klastra muszą być w tej samej podsieci / sieci fizycznej.

- **zmiana maski sieci wszystkich interfejsów komunikacji wewnętrznej klastra**
  Prawdopodobnie jedyną sytuacją, kiedy wystąpi potrzeba zmiany maski sieci wszystkich interfejsów będzie przeniesienie klastra do nowej sieci z inną maską sieci. Aby zmienić maskę sieci należy wyedytować plik `/etc/boottab` w ramdysku w `/boot` a potem uruchomić `ssi-ksync` alby uaktualnić wszyskie pozostałe kopie. Dodatkowo wpisy NETMASK w skrypcie `network-script/ifcfg-eth#` powinny zostać poprawione. Należy zrestartować węzły bez polecenia `mkinitrd`, ponieważ to odtworzyłoby plik `boottab` z maską uruchomionego interfejsu.

- **zmiana bramy dla całego klastra lub pojedynczego węzła**
  Można wprowadzić klauzulę GATEWAY w `/etc/sysconfig/network` i zmiany te będą dotyczyły wszystkich węzłów. Jeśli wymagane są różne bramy dla różnych węzłów, NIE należy umieszczać wpisu w `/etc/sysconfig/network`, ale zamiast tego wstawić ją do `network-scripts/ifcfg-eth#` w każdym `/cluster/node#/etc/sysconfig`.

Dodawanie nowego interfejsu (nie przeznaczonego do komunikacji wewnętrz klastra) do węzła może zostać wykonane przy użyciu skryptu `system-config-network` lub komendą `netconfig` na węźle do którego chcemy dodać nowy interfejs. Wskazówka: wygląda na to, że nie ma strony podręcznika (*man*) dla `netconfig`, dlatego warto skorzystać z polecenia `netconfig --help`. Należy

również zawsze wyszczególnić którego interfejsu dotyczy polecenie przez przełącznik `--device=xxx` (np. `--device=eth1`). W przeciwnym przypadku domyślnie użytym interfejsem będzie `eth0`, który najprawdopodobniej będzie skonfigurowany wcześniej do komunikacji wewnętrznej.

`/etc/clustertab`

Jak widać w powyższych opisach, plik `/etc/clustertab` jest podstawowym plikiem konfiguracyjnym służącym do ustawienia komunikacji wewnętrznej klastra. W pliku tym w kolejności zapisane są:

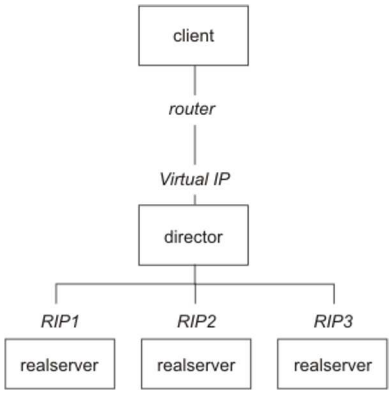
- numer węzła,
- adres IP interfejsu do połączenia wewnętrznego,
- adres MAC interfejsu do połączenia wewnętrznego,
- protokół służący do uruchamiania (*boot*) węzła (zwykle E - Etherboot, alternatywa: P - PXE),
- informacja o tym, czy węzeł może uruchomić pojedynczy proces `init` (jeśli więcej niż jeden węzeł będzie miał włączoną tę opcję - w przypadku awarii następne węzły przejmują funkcjonalność),
- lokalne urządzenie rozruchowe (obligatoryjne dla węzła uruchamiającego `init`).

3 Komunikacja standardowa

Komunikacja sieciowa w przypadku rozwiązania OpenSSI w zasadniczy sposób różni się od klasycznych rozwiązań dla samodzielnych maszyn. Przede wszystkim w przypadku SSI grupa maszyn musi być widziana dla świata zewnętrznego jako jedna, skalowalna i wysoce dostępna maszyna z pojedynczym źbiorczymądresem sieciowym. Jest to jeden z ważniejszych celów SSI z punktu widzenia obsługi sieci. Jest to jeden z powodów, dla których OpenSSI skorzystało z rozwiązania LVS (Linux Virtual Server), który dostarcza takiej funkcjonalności.

LVS zapewnia, że klaster ”udostępniańa zewnątrz pojedynczy adres IP, a połączenia przychodzące są rozkładane pomiędzy maszyny obsługujące odpowiednie usługi przy wykorzystaniu rozmaitych algorytmów (jest to związane z równoważeniem obciążenia sieci - *network load balancing*). Zastosowanie LVS w OpenSSI zostało wzbogacone o bardziej automatyczną konfigurację oraz zwiększenie dostępności.

Zauważyć należy, że z punktu widzenia systemu SSI, ważna jest również widoczność klastra z jego wnętrza oraz widoczność świata zewnętrznego z wnętrza klastra. Chodzi przede wszystkim o to, aby wszyskie urządzenia sieciowe (a przez to i adresy IP) w klastrze wyglądały jak urządzenia lokalne na każdym węźle klastra (LVS zapewnia wszyskim węzłom widoczność jedynie ogólnego adresu IP klastra). Dzięki temu nasłuchiwanie z jakiegokolwiek węzła mogłoby stać się w efekcie nasłuchiwaniem wszyskich interfejsów sieciowych wszyskich węzłów. Dodatkowo, kompletne rozwiązanie SSI pozwalałoby na porozumiewanie się dwóch procesów przez interfejs *loopback*, nawet jeśli wykonywane byłyby na różnych węzłach. Ponadto tablice routingowe powinny być automatycznie synchronizowane tak, że jeśli jakikolwiek węzeł w klastrze może połączyć się z pewnym hostem, wtedy wszyskie węzły powinny mieć możliwość połączenia się z tym samym hostem.



Rysunek 1: Real IP i Virtual IP

Na dzień dzisiejszy w rozwiązaniu OpenSSI została zaimplementowana funkcjonalność LVS wraz z opcją *failover*, czyli przejmowaniem fukcjonalności przez inne węzły w przypadku awarii węzła odpowiedzialnego za pewną usługę.

3.1 CVIP - Cluster Virtual IP

Ideę CVIP oraz LVS pokazana została na Rysunku 1.

CVIP to wirtualny adres IP klastra. Adres ten jest używany przez świat zewnętrzny do połączenia z klastrem. Należy zauważyć, że musi on być różny od zestawu adresów IP wewnątrz klastra, tzn. że CVIP nie może być adresem korzenia (ang. *root node*). Plikiem konfiguracyjnym zawierającym ustawienia dotyczące CVIP jest `/etc/cvip.conf`. Jest to plik w formacie XML i zawiera informacje dotyczące CVIP, węzłów głównych (ang. *director nodes*) oraz prawdziwych serwerach (ang. *real servers*).

Węzły główne przekierowują żądania do usług obsługiwanych przez LVS do odpowiednich węzłów zwanych prawdziwymi serwerami, na których dane usługi są faktycznie uruchomione. Węzły główne mogą dzielić funkcjonalność - mogą być zarówno serwerami głównymi, jak i prawdziwymi.

W przypadku wielu adresów CVIP albo powinny być one dzielone przez wszystkie węzły główne, albo nie powinny być dzielone przez żadne węzły. Oznacza to, że więcej niż jeden CVIP może być skonfigurowany w klastrze pod warunkiem, że będą miały dokładnie te same zestawy węzłów głównych je obsługujących.

3.2 Konfigurowanie LVS/CVIP

Najważniejsze kroki podczas konfigurowania LVS i CVIP to:

1. wybrać adres IP, który będzie służył jako CVIP. Adres ten musi znajdować się w tej samej podsieci co adres sieciowy fizycznego interfejsu, z którego oczekujemy ruchu wchodzącego. (adresem tym może być ten sam adres, który już został przypisany interfejsowi `eth0`, ale klaster zostanie pozbawiony w ten sposób własności *failover*, czyli podstawowej funkcjonalności LVS).

2. uaktualnić plik `/etc/clustername` aby uzupełnić nazwę przypisaną adresowi wybranemu w kroku poprzednim,

3. wyedytować plik `/etc/cvip.conf` - wstawić CVIP, ustawić węzły główne oraz prawdziwe serwery,
4. upewnić się, ze usługa `ha-lvs` jest uruchomiona,
5. oznaczyć porty, na których ma się odbywać *load levelling* (np. przez opcję `setport-weight` skryptu `/etc/init.d/ha-lvs`),
6. zrestartować węzły,
7. Jeśli istnieją węzły z serwerami w sieci wewnętrznej, która nie jest prywatna, należy na nich ustawić drogę powrotną (ang. *route*) do środowiska zewnętrznego, a podana bramka musi mieć włączoną opcję przekazywania IP.
8. Jeśli istnieją węzły z serwerami w sieci wewnętrznej, która jest prywatna, należy włączyć LVS-NAT oraz skonfigurować NAT na węźle głównym. Aby to uczynić należy utworzyć plik /etc/defaults/lvs_routing z zawartością:
   LVS_ROUTING=NAT
   LVS_INTERNAL_GW=¡adresIP¿
   gdzie `adresIP` to adres IP wewnętrznego interfejsu węzła głównego z CVIP.

Aby sprawdzić poprawność działania LVS/CVIP można:

- wydać polecenie `onall ifconfig` - dla węzła głównego w wyniku wystąpi CVIP przypisany odpowiedniemu interfejsowi (np. `eth1`), natomiast pozostałe węzły będą miały przypisany CVIP interfejsowi `lo`,
- jako użytkownik root, wydać polecenie `cat /proc/cluster/lvs` w wyniku którego powinno być widać adres CVIP oraz węzeł główny. Można również poleceniem cat /proc/cluster/ip_vs_portweight sprawdzić dla których portów będzie rozkładane obciążenie,
- jako użytkownik root, wydac polecenie `ipvsadm -L` na węźle głównym aby zobaczyć dla których portów jest rozkładane obiciążenie.

Chociaż CVIP jest dobrym sposobem na dostęp do klastra z zewnątrz, mogą pojawić się problemy przy dostępie ź wewnątrz". Problem nie występuje w przypadku używania LVS-NAT (jak w powyższej konfiguracji). Jeśli jednak do routingu używana jest metoda DR (ang. *direct routing*), występują działania niepożądane. Otóż każdy węzeł ma skonfigurowany adres CVIP na swoim interfejsie zwrotnym (loobpack - lo) aby mógł odbierać przekierowywany ruch sieciowy. Efekt uboczny jest jednak taki, że jeśli wystąpi próba połączenia do adresu CVIP z serwera na takim węźle, żądanie zostanie po prostu wysłane do lokalnego interfejsu lo i de facto nie trafi w ogóle do sieci. Prace nad rozwiązaniem tego problemu trwają.

4 Gniazda

Biorąc pod uwagę fakt, że OpenSSI jest rozwiązaniem klastrowym, w którym rzeczą normalną jest migracja procesów, powstać może pytanie: co się dzieje z gniazdami podczas migracji procesów? Wyobrazić sobie można sytuację, w której takie gniazda migrują razem z procesem. Proces taki powinien wówczas przebiegać następująco:

1. zablokować wszystkie procesy, które próbują dostać się do gniazda tak, aby stan gniazda pozostał niezmieniony w trakcie migracji,

2. dla gniazd sieciowych (czyli prawie wszystkich typów poza `Unix domain`), albo odrzucić przychodzące pakiety bez potwierdzania (tzn. wysyłania odpowiedniego ACK) lub wkładać je do kolejki, która będzie przekierowana do nowej lokalizacji gniazda kiedy proces migracji się zakończy

3. zapakować stan gniazda i stworzyć jego duplikat na nowym węźle (proces ten może być trudny do wykonania w sposób ogólny dla wszystkich rodzajów gniazd)

4. dla gniazd `Unix domain` należy uaktualnić położenie gniazda w klastrowym serwerze nazw oraz uaktualnić wszystkie gniazda, które już są połączone z gniazdem migrującym,

5. dla gniazd połączonych z CVIP, uaktualnić tablicę HA-LVS, wstawiając nową pozycję gniazda,

6. wysłać nowe położenie gniazda do wszystkich procesów, które miały to gniazdo otwarte,

7. jeżeli do przechowywania nadchodzących pakietów używany był mechanizm kolejki, przekazać jej zawartość do nowego położenia gniazda,

8. wznowić wszystkie zablokowane procesy,

9. usunąć gniazdo na poprzednim węźle.

Jak widać jest to proces złożony i skomplikowany. Implementacja takiego mechanizmu mogłaby negatywnie odbić się na wydajności (chociażby przez blokowanie procesów). Dlatego też nie została podjęta próba włączenia go do rozwiązania OpenSSI.

4.1 Wiązanie gniazd

Mimo iż operacja migracji gniazd nie została zaimplementowana, istnieć musi mechanizm wspierający operację na gniazdach podczas przenoszenia procesu pomiędzy węzłami klastra. Odbywa się to poprzez tworzenie fałszywego pliku (ang. *dummy file*), *inode* oraz struktur gniazda na nowym węźle oraz zainstalowaniu wektora zdalnych operacji na tych strukturach. Operacje te mają informacje jak znaleźć prawdziwy obiekt na innym (pierwotnym) węźle, dostarczyć odpowiednim operacjom na tym obiekcie argumenty oraz odebrać wyniki. Cecha ta nie jest realizacją idei globalnych gniazd (ang. *clusterwide sockets*), ale pozwala procesom i gniazdom na 'rozproszenie' w klastrze. W trakcie procesu migracji gniazdo jest zatem cały czas związane z tym samym węzłem, na którym zostało utworzone. Każda operacja na nim (np. `read()` / `write()`) odbywająca się na nowym węźle jest przekazywana do starego węzła, z którym gniazdo zostało związane (`bind()`).

Odrębnym zagadnieniem jest sensowność i użyteczność migrujących gniazd. Można sobie wyobrazić sytuację, w której gniazdo - związane z adresem IP pewnego węzła - przenoszone jest do nowego węzła. Jeśli pierwotny węzeł ulegnie awarii, gniazdo straci swoje połączenie. W tym świetle wydaje się, że rozpatrywać można jedynie migrowanie tych gniazd, które związane są z wirtualnym IP klastra, czyli CVIP (w tym przypadku utrata połączenia nie będzie miała miejsca ze względu na cechę *failover*).

4.2 Implementacja

[*Uwaga: aby w pełni zrozumieć przedstawione tu zmiany, należy posiadać przynajmniej podstawową wiedzę na temat organizacji jądra systemu Linux i jego warstwy sieciowej. Pomocna może być lektura http://www.tldp.org/LDP/tlk/net/net.html*]

Z oczywistych względów interfejs gniazd BSD (ang. *BSD sockets*) nie został zmieniony - zmiany zaszły w strukturze wewnętrznej i działaniu. Zostało to jednak ukryte pod powierzchnią interfejsu.

Pierwsza, dość wyraźna zmiana, zaszła w strukturze `struct socket` warstwy gniazd BSD (znaleźć ją można w pliku `include/linux/net.h`). Struktura ta jest podstawowym elementem wewnętrznej implementacji gniazd BSD. Zawiera m.in. wskaźnik na odpowiedni węzeł (ang. *inode*) w systemie plików, stan gniazda oraz wskaźnik do struktury proto_ops, która z kolei zawiera listę wskazań na odpowiednie funkcje operujące na danym typie gniazda. W stosunku do swojego pierwowzoru, w rozwiązaniu OpenSSI zyskała dwa nowe pola:

```
unsigned long ssi˙rfb˙id;
clusternode˙t ssi˙rfb˙svr;
```

(rfb - remote file block)
Pierwsze pole `ssi˙rfb˙id` jest inicjowane w momencie akceptacji połączenia przychodzącego w funkcji:

```
int rmtsock˙accept(struct socket *sock, struct socket *new-
sock, int flags)
```

(plik `cluster/ssi/util/rmtsock.c`)

Wartość, która zostaje mu nadana jest wartością pochodzącą z generatora określonego w pliku `include/cluster/ssi/util/unum.h` i jest wartością unikalną w zakresie swojej domeny gniazd. Z kolei `ssi˙rfb˙srv` jest inicjowane numerem serwera z przychodzącej struktury `socket`. Na tej podstawie tworzone jest nowe gniazdo.

Wspomnieć można, że część struktur używanych w tym procesie została utworzona z właściwym twórcom jądra humorem:

```
u32 rfb˙magic; /* hocus pocus */
```

Kolejną ważną zmianą, na którą należy zwrócić są funkcje obsługi gniazd BSD. Wszystkie funkcje z przedrostkiem `rmtsock` zebrane zostały w standardową strukturę typu `proto˙ops` i nazwane `rmtsock˙ops`. Struktura ta następnie jest podłączana do wskaźnika `proto˙ops` struktury `socket` dzięki czemu nowo wprowadzone funkcje mogą obsługiwać komunikację sieciową. Dodatkowo w pliku `net/socket.c` w stosunku do klasycznego jądra Linuksa, w strukturze typu `super˙operations` w inicjalizatorze podmieniona została procedura obsługująca odczytywanie węzłów (ang. *inode*) związanych z gniazdami na procedurę zdefiniowaną w `cluster/ssi/util/rmtsock.c` (w tym samym pliku znajduje się definicja `rmtsock˙ops`). Struktura `super-˙operations` jest częścią systemu plików (`include/linux/fs.h`) i zawiera wskaźniki do wielu funkcji, które są dostępnymi operacjami na pliku. Ponieważ gniazdo reprezentowane jest w systemie plików, funkcje zawarte w tej strukturze są

de facto funkcjami na gnieździe. Jeśli przypomnimy sobie, że OpenSSI korzysta z fałszywych plików (ang. *dummy file*) podczas migracji procesów, przedefiniowanie takich operacji okazuje się być uzasadnione.

Ostrzeżenie
Warto zwrócić uwagę na błąd związany z komunikacją sieciową, który od ok. 1,5 roku nie został naprawiony (nawet w wersji 1.9.0). Otóż funkcja `rcvmsg` może spowodować panikę jądra (ang. *kernel panic*) węzła na którym zostanie uruchomiona. Błąd powodowany jest m.in. w przypadku przekazania do funkcji `NULL` zamiast typu wiadomości. Odpowiadająca za odbieranie wiadomości funkcja poziomu gniazd BSD to `rmtsock˙recvmsg` znajdująca się w pliku `cluster/ssi/util/rmtsock.c`.

Literatura